Object-Oriented Programming I (CSE 121)
Week # 08
(Encapsulation)

Objective(s)

In this lecture, you will learn:

1. What is Encapsulation?
2. Access specifiers or modifiers
3. How encapsulation is achieved in a class?
4. Encapsulation examples

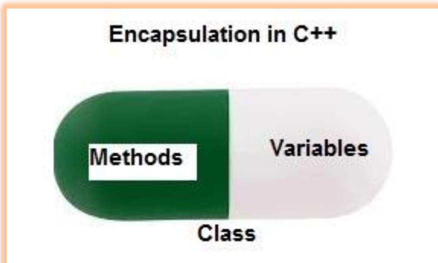
Encapsulation

Object-Oriented Programming (OOP) has four core concepts which are listed below:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Encapsulation

- **Encapsulation** is a process of combining data members and functions in a single unit called class.
- In other words, **Encapsulation** is an OOP concept which binding together the data and the functions that manipulates those data.



class is an example of encapsulation. If we have created some class and have data members (variables) and member functions (methods) then it is an example of encapsulation.

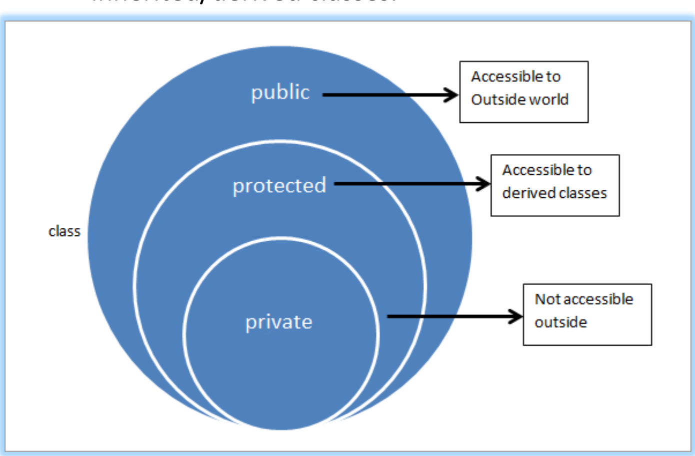
- Encapsulation is to prevent the access to the data directly, the access to them is provided through the functions of the class. It is one of the popular features of Object-Oriented Programming (OOP) that helps in **data hiding**.

Data Hiding: It is an object-oriented programming concept that hide data from user so that accidental changes can be avoided.

Access Specifiers or Modifiers

We can implement **encapsulation** by using **access specifiers**. They give the programmer the control on what data or functions are to be made visible to the user and what is kept a secret. In C++, there are three access specifiers:

- public - members are accessible from outside the class
- private - members cannot be accessed (or viewed) from outside the class
- protected - members cannot be accessed from outside the class, however, they can be accessed in inherited/derived classes.



Access Specifiers	Within same class	In derived class	Outside the class
public	Yes	Yes	Yes
private	Yes	No	No
protected	Yes	Yes	No

Note: Each of these access specifiers is used to implement Encapsulation. By default, all data members and member function are made **private** by the compiler.

How Encapsulation is achieved in a class

To do this:

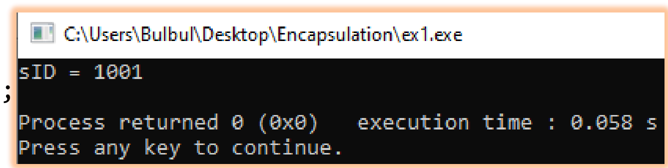
1. Make all the data members **private**.
2. Create **public** setter and getter functions for each data member in such a way that the set function set the value of data member and get function get the value of data member.

Encapsulation Example in C++

Example 1: A simple example of encapsulation.

```
//Encapsulation implemented using class and access modifiers.
#include <iostream>
using namespace std;

class Student{
private:
    int sID;
public:
    //setter
    void setID(int a){ //function to set variable x
        sID = a;
    }
    //getter
    int getID(){ //function to return value of variable x
        return sID;
    }
};
int main(){
    Student obj; //create an object of Student
    obj.setID(1001);
    cout << "sID = " << obj.getID() << endl;
    return 0;
}
```



Example explained

- The sID attribute is private, which have restricted access.
- The public setID() method takes a parameter (a) and assigns it to the sID attribute (sID = a).
- The public getID() method returns the value of the private sID attribute.
- Inside main(), we create an object of the Student class. Now we can use the setID() method to set the value of the private attribute to 1001. Then we call the getID() method on the object to return the value.

Why Encapsulation?

- Encapsulation is one of the most important features of OOP as it provides us a way to hide the data. It is considered good practice to declare your class attributes as private (as often as you can).
- Increased security of data.
- Encapsulated classes reduce complexity.

Example 2: Another example of encapsulation with 1 setter and 3 getter functions.

```cpp
#include <iostream>
using namespace std;

class Student {    // The class
  private:          // Access specifier
    int sID;        // Attribute (int variable)
    string sName;   // Attribute (string variable)
    double gpa;     // Attribute (double variable)
  public:
    //setter function: setValue()
    void setValue(int x, string y, double z)
    {
        sID = x;
        sName = y;
        gpa = z;
    }
    //getter functions: getID(), getName() and getGPA()
    int getID()
    {
        return sID;
    }
    string getName()
    {
        return sName;
    }
    double getGPA()
    {
        return gpa;
    }
};
int main()
{
    Student obj;
    obj.setValue(1001,"Bulbul Ahamed",3.95);
    cout << obj.getID() << endl;
    cout << obj.getName() << endl;
    cout << obj.getGPA() << endl;
    return 0;
}
```

C:\Users\Bulbul\Desktop\Encapsulation\ex2.exe

```
1001
Bulbul Ahamed
3.95

Process returned 0 (0x0)   execution time : 0.081 s
Press any key to continue.
```

In the program above, we have bundled three member variables along with the getter and setter methods into a class. So, it is an example of encapsulation.

We have declared three variables i.e. **sID**, **sName** and **gpa** as private variables so that they are not accessible to the outside world. They are only accessible to the functions that we have declared as public. Thus, we have hidden data members as private variables in a class.

Let us take another example to better understand Encapsulation in C++.

Example 3: Example of encapsulation.

```cpp
#include <iostream>
using namespace std;

class Account{
private:
    double balance;
public:
    Account(double amount){ //constructor
       balance = amount;
    }
    ~Account(){ //destructor
        cout << "Destructor is called\n";
    }
    //setter method/function
    void deposit(double amount){
        if(amount > 0){
            balance += amount;
        }
    }
    //setter method/function
    void withdraw(double amount){
        if(amount > 0  && balance >= amount){
            balance -= amount;
        }
    }
    //getter method/function
    double getBalance(){
        return balance;
    }
};

int main(){
    Account a1(5000);  //Constructor called automatically when object is created
    cout << "Initial deposit: " << a1.getBalance() << endl;

    a1.deposit(10000);  //deposit function called
    cout << "After 2nd deposit: " << a1.getBalance() << endl;

    a1.deposit(7000);  //deposit function called
    cout << "After 3rd deposit: " << a1.getBalance() << endl;

    a1.withdraw(2000);  //withdraw function called
    cout << "Current Balance: " << a1.getBalance() << endl;
    return 0;
}
```

In this example, we have a class **Account** that bundles account **data** and all the **functions** that operate on this data into a single class Account. In the **main function**, we can create an **object** of this class and access functions to get the desired information.

C:\Users\Bulbul\Desktop\Encapsulation\ex3.exe

```
Initial deposit: 5000
After 2nd deposit: 15000
After 3rd deposit: 22000
Current Balance: 20000
Destructor is called

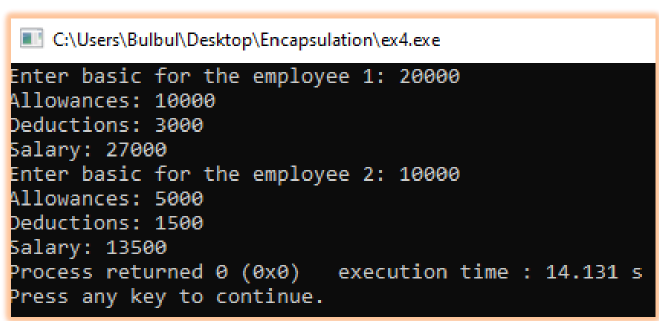
Process returned 0 (0x0)   execution time : 0.107 s
Press any key to continue.
```

Example 4: Another example of encapsulation.

```cpp
#include <iostream>
#include <string>
using namespace std;

//Accounts class: includes salary info for a particular employee
class Accounts{
private:
    int empId;
    double salary, basic, allowances, deductions;
public:
    //constructor
    Accounts(int empId){
        this -> empId = empId;
    }
    //read salary info
    void readEmployeeInfo(int empId){
        cout << "Enter basic for the employee " << empId <<": ";
        cin >> basic;
        cout << "Allowances: ";
        cin >> allowances;
        cout << "Deductions: ";
        cin >> deductions;
    }
    //calculate salary
    double calculateSalary(){
        salary = basic+ allowances - deductions;
        return salary;
    }
    //display details
    void display(){
        salary = calculateSalary();
        cout << "Salary: " << salary;
    }
};
int main(){
    Accounts acc(1);
    acc.readEmployeeInfo(1);
    acc.display();
    cout << endl;
    Accounts acc2(2);
    acc2.readEmployeeInfo(2);
    acc2.display();
    return 0;
}
```

In this example, we have a class **Accounts** that bundles account **data** and all the **functions** that operate on this data into a single class Account. In the **main function**, we can create **objects** of this class and access functions to get the desired information. Here, necessary inputs are taken from user.



```
C:\Users\Bulbul\Desktop\Encapsulation\ex4.exe
Enter basic for the employee 1: 20000
Allowances: 10000
Deductions: 3000
Salary: 27000
Enter basic for the employee 2: 10000
Allowances: 5000
Deductions: 1500
Salary: 13500
Process returned 0 (0x0)   execution time : 14.131 s
Press any key to continue.
```

………………………………